27503

INTEL INVENTION DISCLOSURE

ATTORNEY-CLIENT PRIVILEGED COMMUNICATION
located at http://legal.intel.com/patent/index.htm

DATE: [redacted] **ARCHITECURE/EPG/MRL/UAL**

It is important to provide accurate and detailed information on this form. The information will be used to evaluate your invention for possible filing as a patent application. **Invention Disclosure forms MUST be sent electronically via email to your manager/supervisor who should then forward with their approval to our email account "invention disclosure submission."** If you have any questions, please call **8-264-0444.**

Last Name: **Brown**	First Name: **Jeffery**		M.I. **A**
Intel Phone Number: **408-765-7642**	Intel Fax Number: **408-653-8511**		Mailstop: **SC12-303**
[redacted]			[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]			
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		[redacted]
[redacted]	[redacted]	[redacted]	[redacted]

Last Name: **Wang**	First Name: **Hong**		M.I.
Intel Phone Number: **408-653-7075**	Intel Fax Number: **408-653-8511**		Mailstop: **SC12-303**
[redacted]			[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
Home Address: [redacted] SAN JOSE, CA 95157			
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		[redacted]
[redacted]	[redacted]	[redacted]	[redacted]

Last Name: **Hammarlund**	First Name: **Per**		M.I.
Intel Phone Number: **(503) 712-2891**	Intel Fax Number: **(503) 712-2891**		Mailstop: **JF4-3FL-J7**
[redacted]			[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
Home Address: [redacted] HILLSBORO, OR 97124			
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
Last Name: **Shen**	First Name: **John**		M.I.
Intel Phone Number: **408-765-5671**_	Intel Fax Number: **408-653-8511**		Mailstop: **SC12-303**
[redacted]			[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
Home Address: [redacted] SAN JOSE, CA 95121			
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
Last Name: **Chrysos**	First Name: **George**		M.I.
Intel Phone Number: **508 841-1641**	Intel Fax Number: **508 841-1642**		Mailstop: **MC-P25**
[redacted]			[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]			

[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		
[redacted]	[redacted]	[redacted]	[redacted]

Last Name: **Wang**	First Name: **Perry**		M.I.
Intel Phone Number: **408-765-4442**	Intel Fax Number: **408-765-4442**		Mailstop: **SC12-303**
[redacted]			[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
Home Address: [redacted] SAN JOSE, CA 95133			
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		[redacted]
[redacted]	[redacted]	[redacted]	[redacted]

Last Name: **Orenstein**	First Name: **Doron**		M.I.
Intel Phone Number: **465-5732**	Intel Fax Number: **465-5732**		Mailstop: **IS-3M1**
[redacted]			[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
Home Address: [redacted]; HAIFA, ISRAEL			
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		[redacted]
[redacted]	[redacted]	[redacted]	[redacted]

Last Name: **Liao**	First Name: **Shih-wei**		M.I. M
Intel Phone Number: **408-653-4954**	Intel Fax Number: **408-653-4954**		Mailstop: **SC12-303**
[redacted]			[redacted]
[redacted]	[redacted]	[redacted]	[redacted]
Home Address [redacted] SAN JOSE, CA 95129			
[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]		[redacted]
[redacted]	[redacted]	[redacted]	[redacted]

(PROVIDE SAME INFORMATION AS ABOVE FOR EACH ADDITIONAL INVENTOR)

2. Title of Invention:
Methodology and Apparatus for Supporting/Facilitating Affinity-guided Speculative Helper Threads in CMP Processors

3. What technology/product/process (code name) does your invention relate to (be specific if you can)
Our techniques improve the performance of co-operative multithreading on CMP processors, where data flow between threads is predictable. The case of Speculative Precomputation and its "helper threads" was our motivating example.

4. Include several key words to describe the technology area of the invention in addition to # 3 above:
CMP multithreading, speculative precomputation, helper thread

5. Stage of development (i.e. % complete, simulations done, test chips if any, etc.):
<u>Simulation done</u>

6a. Has a description of your invention been (or planned to be) published outside of Intel:

NO

If YES, was the manuscript submitted for pre-publication approval through the Author Incentive Program:

If YES, please identify the publication and the date published:

6b. Has your invention been used/sold or planned to be used/sold by Intel or others?

If YES, date it was sold or will be sold:

NO

6c. Does this invention relate to technology that is or will be covered by a SIG (special interest group)/standard or specification? **NO**

If YES, name of SIG/standard/specification:

6d. If the invention is embodied in a semiconductor device, actual or anticipated date of tapeout?

6e. If the invention is software, actual or anticipated date of any beta tests outside Intel:

7. Was the invention conceived or constructed in collaboration with anyone other than an Intel blue badge employee **NO**
or in performance of a project involving entities other than Intel (e.g. government, other companies, universities
or consortia)? **NO**: If YES, name of individual or entity:

8. Is this invention related to any other invention disclosure that you have recently submitted? If so, please give the title and
inventors:

****

PLEASE READ AND FOLLOW THE DIRECTIONS ON HOW TO WRITE A DESCRIPTION OF YOUR INVENTION

Try to limit your description to 2-3 pages Do NOT attach a presentation, white paper, or specification ANSWER ALL OF THE QUESTIONS BELOW

Please provide a description of the invention and include the following information:

1. **Describe in detail what the components of the invention are and how the invention works.**

2. **Describe advantage(s) of your invention over what is currently being done.**

3. **You MUST include at least one figure illustrating the invention. If the invention relates to software, include a flowchart or pseudo-code representation of the algorithm.**

4. **Value of your invention to Intel (how will it be used?).**

5. **Explain how your invention is novel. If the technology itself is not new explain what makes it different.**

6. **Identify the closest or most pertinent prior art that you are aware of.**

7. **Who is likely to want to use this invention or infringe the patent if one is obtained and how would infringement be detected?**

HAVE YOUR SUPERVISOR READ AND FORWARD IT ELECTRONICALLY

VIA E-MAIL TO "INVENTION DISCLOSURE SUBMISSION"

DATE: ________________ SUPERVISOR: ____

BY APPROVING, I (SUPERVISOR) ACKNOWLEDGE THAT I HAVE READ AND UNDERSTAND THIS
DISCLOSURE, AND RECOMMEND THAT THE HONORARIUM BE PAID

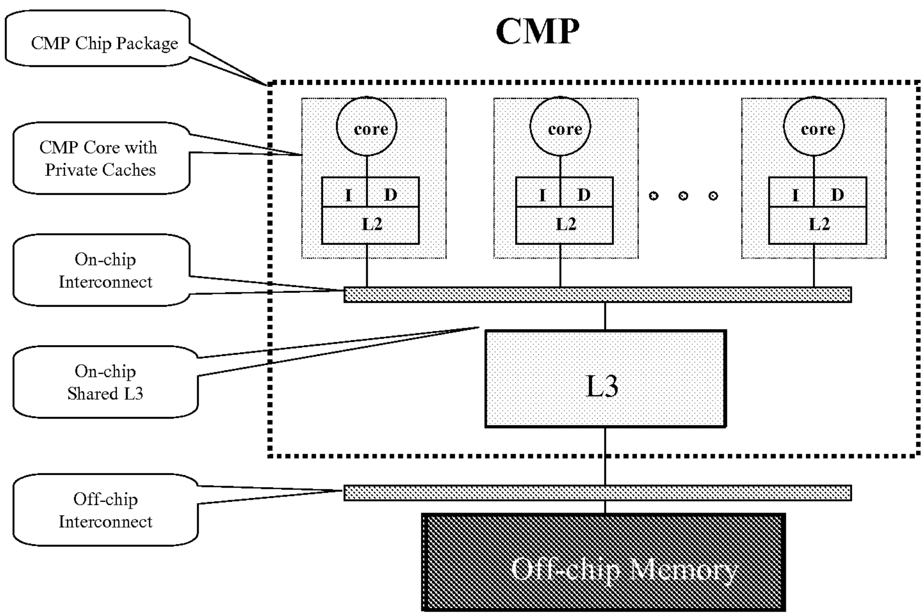
Methodology and Apparatus for Facilitating Affinity-guided Speculative Helper Threads in CMP Processors

Jeffery A. Brown[1]
Hong Wang[1]
Per Hammarlund[2]
George Chrysos[?]
Perry Wang[1]
John Shen[1]
Doron Orenstein[1]
Steve Shih-wei Liao[1]

[1] Microprocessor Research Labs, Intel Labs
[2] Desktop Product Group, Intel Corporation
[?] MMDC, EPG, Intel Corporation

1. Executive Summary

This disclosure introduces a set of 5 novel methods to improve inter-core communication on CMP processors, such as the one depicted below,



in order to speed up single-threaded application performance via "speculative helper threading" techniques such as Speculative Precomputation.

In Speculative Precomputation [7, 8, 9, 10, 11, 12, 13, 14, 15], the otherwise idle thread resources are used to pre-fetch critical data for the single-threaded main thread thread that is being helped. These helper-induced pre-fetches speed up the main thread by helping to ensure that critical data are in cache by the time they are needed by the main thread, by exploiting the presence

of affinity of resource sharing, in particular, shared data caches. SP was originally conceived for SMT processor models, where data caches are shared at the closest levels of the memory hierarchy to the processor core; however, this is not the case for CMP processors, where the sharing occurs farther up (and away from the core the pipeline) in the hierarchy, and communications from helper threads to the main thread potentially suffer from higher latency, reducing the performance benefit from SP, should the helper threads and the main thread be running from different CMP cores.

To address the new challenges, in this disclosure, we propose 5 techniques to improve inter-core communication to support speculative helper threads on CMP processors.

1. First is a scheme for affinity-based "return data multicast" between main core and its helper cores: when a miss from one core's cache is serviced by the shared cache, the result is multicast to all cores in the affinity group of interest and the data will be injected into all cores' private data caches. This not only allows the effects of the helper thread prefetches to be realized much earlier in the main thread's core but also allows helper cores' private cache to be warmed up for intermediate data that helper thread would need, therefore speeding up helper threads and in turn enhancing the prefetch timeliness since the helper threads can run further ahead and minimize stalls due to cold misses in helper core.

2. Second is a scheme for "peer-2-peer/point-2-point cross-feed" between a pair of cores of interest (main-helper, helper-helper): when a miss is sent from one core to the shared cache, the helper cores may service the request from their own caches. This entails little if any at all change to and maximally leverage most existing cache coherence schemes. Unlike multicast (as in 1), this is an on-demand supply of missing data for the main core from the helper core's private data cache.

3. The third scheme is the "directed prefetch" scheme, where pre-fetches issued by a helper core are returned only into the main core, assuming point-to-point kind of interconnection network.

4. The fourth scheme is adaptive combo/hybrid schemes of the above schemes. The adaptation can be enabled via dynamic monitoring of interconnection network bandwidth consumption or via programming logics in helper threads by compiler, user or OS, statically or dynamically.

5. The fifth scheme is to augment cache coherence protocol to handle a special form of speculative instruction that is called pseudo-coherent safe-store, to support helper threads involving data structure mutation, which therefore needs to perform stores to ensure correct prefetch. Another use is helper core enabled instruction cache prefetch, or prescient instruction prefetch [16] where it is inevitable for a helper thread to encounter store during its speculative execution of a future portion of the main thread program, which likely will have store.

   Unlike speculative store that has been generally discussed in the context of traditional speculative multithreading and geared towards value reuse [1, 2, 3, 4, 5], safe-store is introduced in a separate disclosure [6] for SMT as a means to allow speculative store from

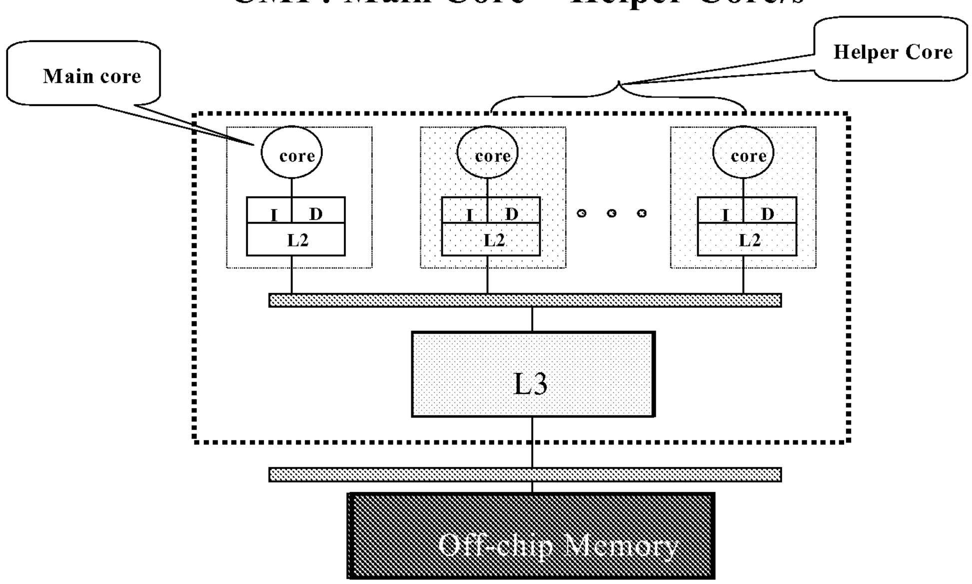
speculative thread to be written in the lower level caches and only allow a load from the speculative thread to use the data as valid, while treating the data as invalid for load from the non-speculative thread. In addition, safe-store is prevented from write-back to memory via similar though simpler mechanism.

The pseudo-coherent safe-store support in augmented cache coherence protocol allows speculative helper thread to do speculative stores in their local private cache and the data block of concern can be used by the speculative helper threads locally or remotely (for potentially different usage), while still prohibiting the safe-store from being written back into the memory. Likely the safe-store for SMT, pseudo-coherent safe-store support enables helper threads involving data structure mutation to observe memory dependency between store and load in the helper thread and ensure accuracy of prefetch.

These schemes can be realized via a set of embodiments including pure microarchitectural and platform topological means, or pure architectural/software means, or hybrid of both software and hardware means. We will further elaborate the details throughout the rest of this disclosure. In addition, we will present simulation results to quantify the tradeoffs of these schemes and contrast CMP approaches with SMT approaches.

Note: for descriptive convenience without confusion, throughout the rest of the disclosure, we will refer cache layer closer to the core pipeline as "*lower*" level cache and layer that are far from the core and closer to memory as "*higher*" level cache.

CMP: Main Core + Helper Core/s



2. Motivation

The baseline SP-on-CMP configuration is to run main thread on a core and run its helper threads on distinct cores, which are called helper cores, as depicted as follows.

By default, the helper threads, if performed timely, can help the main thread prefetch and warm up the shared higher level cache. In the example configuration, the helper threads will help the main thread to warm up L3 cache.

Compared to the SMT models on which SP was originally conceived, CMP machines, as depicted as follows, have significantly longer and potentially non-uniform communication delays between threads respectively running on different cores. Typically, SMT models allow threads to communicate through a shared first-level data cache, but CMP models have private lower level caches and only share memory resources at higher levels of the memory hierarchy. While CMP models may allow for more parallelism due to less resource contention, they could also decrease the performance gains from SP techniques, due to the increased communication latency between threads/cores and due to reduced degree in cache sharing. This motivates discovery of new techniques to hide or circumvent communication latency and ensure efficacy of SP, in terms of timeliness and correctness of the SP-induced prefetches.

3. Overview of Key Ideas and Claims

In this invention, we propose 5 simple techniques to mitigate the impact of cross-core communication latencies on CMP to facilitate SP helper threads. The essential underlying idea is to get the blocks prefetched by the helper threads to a place where the main thread can access them quickly. We will first describe the mechanisms and then provide results of detailed simulation studies to quantify the merits of these innovations.

Given the very high degree of memory-read sharing between threads in SP prefetching, there is a very high likelihood that blocks requested by one thread will later be requested by at least one other among peer cores. Since the helper threads are running prefetch slices, i.e., a subset of the main program, and out ahead of the main thread, as long as they are *effective* and *on-track* (in terms of *correctness* in generated prefetch addresses and *timeliness* of issued prefetch instructions) then the main thread will eventually request the same blocks used by the helper thread. Conversely, during their computation, SP threads often require data from blocks are already in use by the main thread or other SP threads. Each of these schemes exploits the high degree of address commonality between threads/cores in the SP-on-CMP environment.

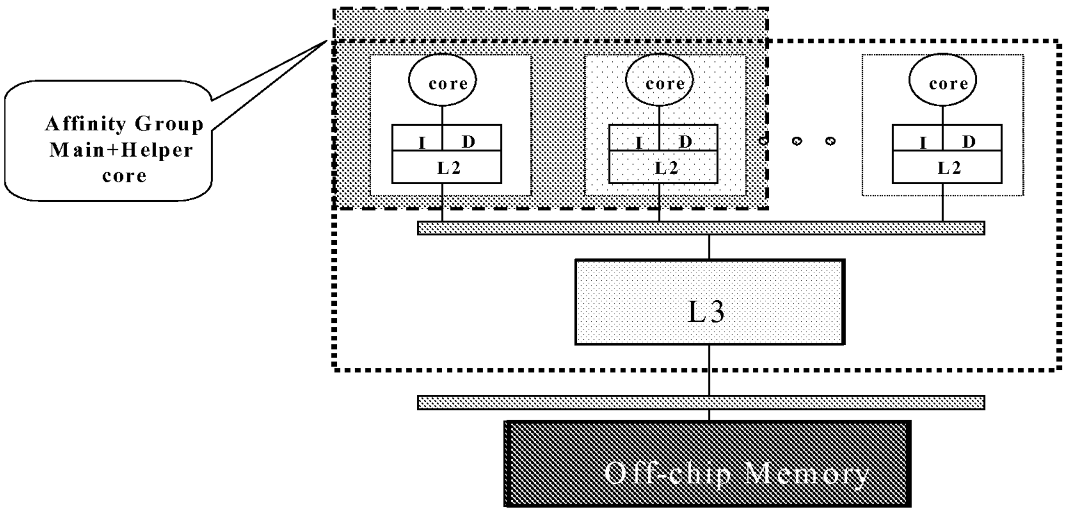
3.1. Broadcasting/multicasting Shared Cache Returns

In this scheme, when the higher level shared cache (such as shared L3) services a miss from one core's lower level private cache (such as private L2), it broadcasts that result data return to all helper cores, which inject that block into their private caches. The unsolicited cache data returns can be propagated down to the lowest levels of cache, thus potentially achieving prefetch to further help main thread reduce the latency of near-future uses.

In general, on a given CMP, based on topological affinity (such as adjacency and hierarchy of interprocessor communication), a subset of CMP cores can be designated to form affinity-group/gang/team/neigborhood, amongst which one core is assigned to run the main thread while the rest run helper threads. Instead of a simple broadcast to all cores on CMP, a particular design

could selectively transmit the cache return to the subset of cores in the affinity gang in a multicast fashion.

SP on CMP: Affinity Group: Main + Helper Core



For small-scale CMP designs with few cores and a bus-like interconnect, only minimal additional resources would be required to adapt snoop schemes to support this multicast scheme. On large scale CMP design, multicasting is also mandatory in the interprocessor communication protocol, and its adapted use for a programmable or topologically fixed affinity-gang organization for helper thread is straightforward. And simple programmability can be introduced to design programmable mask-based variable scope to group core subset and enable respective multicasting.

So for broader claims, the mechanism described above allow multiple distinct affinity-gangs to concurrently interleave cores in CMP to run (main+helpers) multiprograms, each having distinct multicasting scope.

Performance-wise, this multicasting based prefetch scheme can help reduce miss rate on private cache, thus improving performance. The potential drawback is increased usage of bandwidth (i.e. traffic) to private caches. However, the traffic increase is justified if the threads of concern otherwise would incur frequent long-latency memory stall and underutilize the available bandwidth on the private cache, should no helper threads/cores be used. More quantitative evaluation will be presented later that highlights the advantage of this scheme, fundamentally due to the address commonality among threads of affinity gangs.

3.2. Just-in-time On-demand Peer Cache Cross-Feeding

In this scheme, when one core's private cache incurs a miss to a shared cache, other cores' private caches opportunistically return the requested data if they have it available. This would require zero or at most minimal changes to pre-existing cache coherence support.

The essential idea is to effectively treat a load miss off the one core, regardless its original encoding (say regular LD rather than LD.acq denoting load on coherent shared data), as a load on shared coherent data (i.e. semantically equivalent to turning any load missing off core into a coherent LD.acq), thereby tripping coherence protocol such that a peer core can potentially supply data block from its private cache, instead of being serviced by the next level cache whose access latency will be longer than coherence protocol engendered inter-peer cache transfer.

On CMP implementations that do **not** support return data multicasting (as described in Section 3.2), this on-demand cross-feeding scheme can help main thread performance by effectively reducing the penalty of on-core cache misses (in contrast, multicasting scheme help reduce on-core cache miss rate).

The effect of turning a load miss off from the core to a coherent load can be achieved via software or hardware or hybrid. In the simpler software scheme, any helper thread, upon generation (online or offline) can be explicitly encoded as coherent load to trip coherence protocol. A better approach would be to engineer CMP core design to use help mode as hint to turn miss request to off-core caches into a request to shared data to effectively make the return data that is eventually brought into the private cache in share state, e.g. should MESI protocol is used. Afterwards, if another peer core incurs private cache miss on the same address and the load request is turned coherent load (say LD.acq), the current core can supply the block in share state to that peer core, hopefully faster than otherwise would be serviced from the next level shared cache.

It is important to note that most if not all helper threads in SP by design do NOT contain stores. Therefore, the helper thread's private data cache unlikely will carry block with exclusive or modified state for data that are prefetched to helper core's private cache and will be used by the main thread.

This effectively means helper core is effectively using the capacity of its private data cache to help the main core *hoard* future data references. In contrast, multicasting schemes in Section 3.2 implies the helper core eagerly help the main core *update* its private cache, thus more aggressive and timely than hoarding on the side as in peer-feeding described here. This phenomenon will be quantified in later presentation of simulation results.

Another perspective on this scheme is to consider helper thread as effectively precomputing not only future data address for prefetch but also a special microarchitectural state, the data block state ownership, an attribute of cache coherence protocol, and consequently further facilitating the effectiveness of the prefetch. This effectively implies a form of LD *forging* for those loads whose original encoding is not necessarily meant for coherent shared data access.

3.3. Directed Prefetching

In this scheme, prefetch instructions from one core may be returned unsolicited into another core. In particular, prefetches issued by a helper thread are injected into the core where the main thread is running, instead of being returned to the helper core. Such a prefetch would bypass the private caches on the helper core, requesting directly from the low level shared cache. This would not require the ability for the core interconnect to broadcast results, or the potential complexity of a

multicast implementation; the prefetch would simply have its "return address" *forged* to indicate that it should be returned to the target core.

This mechanism can be considered a simplified affinity-based scheme as described in Section 3.2. Or rather this is uni-casting mechanism, where a helper is producer core and the main is the consumer core. The doctoring of return data address can be done via manipulating the routing or navigation mask, which is usually programmable in most modern link-based interconnection network. In other words, the doctoring of return data target core itself could be considered part of precomputation task.

This SP paradigm effectively not only precompute prefetch address and perform prefetch, but also precomputes microarchitectural states on return data navigation and routing mask. Comparing to multicasting, the unicast mask can be considered a special case.

3.4. Bandwidth Adaptive Prefetching

A class of hybrid schemes can derived based on the schemes described above. Broadcast, multicast, unicast can be used in combination of just-in-time on-demand peer-to-peer coherence protocol enabled inter-feeding schemes. In particular, the adaptation can be made based on bandwidth consumption and scale of the CMP interconnection network. In small scale CMP or small scope of affinity gang, multicasting or broadcasting schemes can be used while for topologically remote ganging cores, coherence protocol based inter-peer feeding and unicasting can be more efficient.

Another level of adaptation can be provided by software based on the workload characteristics in compiler or OS or programmer, which have better understanding on the delinquent loads of interest and how particular helper thread is written (statically by user or compiler) or scheduled or physically/topologically mapped (by OS or lower-level scheduler which could be in the app at user level itself).

3.5. Pseudo-coherent Safe-Store Support for Helper on CMP

In some benchmark such as VPR, the dynamic data structure, whose access frequently incurs cache misses, also incurs frequent data structure mutation, e.g., rearrangement and shuffling of nodes in a linked list. The pointer update requires store operation. Ignoring such mutation updates, a helper thread that only chase pointers via tracking loads, will be oblivious of data structure change and eventually will run off course and astray, therefore less effective if not entirely detrimental.

Unlike speculative store that has been generally discussed in the context of traditional speculative multithreading and geared towards value reuse [1, 2, 3, 4, 5], safe-store is introduced in a separate disclosure [6] for SMT as a means to allow speculative store from speculative thread to be written in the lower level caches and only allow a load from the speculative thread to use the data as valid, while treating the data as invalid for load from the non-speculative thread. In addition, safe-store is prevented from write-back to memory via similar though simpler mechanism.

On CMP, each helper core's private cache can be easily implemented to support safe-store as described in [6]. Therefore, all local store and load dependency are observed in the private cache, while none of the stores from helper core are allowed to be observed outside the core.

However, there is benefit to expose safe-store amongst peers in an affinity gang on CMP. For example, the safe-store on common address can be used effectively by one helper thread to terminate another, which otherwise would likely run astray on a stale data structure.

The pseudo-coherent safe-store support in augmented cache coherence protocol to augment the I-state (invalid) of data block with potentially 1 more bit for safe-store bit (effectively indicating *exclusive* state among helpers) on top of the MESI protocol in similar fashion that safe-store bit is used as extension of invalid bit in regular cache design.

Each safe-store on a given helper core will mark its local copy *exclusive* yet safe-store, while engendering invalidation on copies of the safe-store data in other helper cores. However, such block, albeit considered valid for loads from helper thread, will be considered invalid by the non-speculative thread. Like safe-store cache in SMT in [6], when a safe-store line is displaced, it will NOT be written back into higher level cache that does not support safe-store, nor will it be written into memory or exposed to other packages or cache layers/clusters that do not recognize safe-store or pseudo-coherence protocol.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

4.3. Performance Results and Analysis

The results are enumerated as follows. The benefits of the techniques described the section 3.1, 3.2, 3.4. are clearly self-evident in terms of effectiveness in helping achieving higher performance beyond the default naïve CMP+SP configuration like configuration 3 and 4.

Some interesting observations can be highlighted as follows

a. Just as in SMT, SP is no panacea to address all illness. For compute intensive benchmarks [redacted] suffering more instruction cache miss like VPR, applying SP targeting data cache prefetching does not bring any benefit. But unlike SMT, for GZIP or VPR, the helper thread does not slow down the main thread. The reason is that GZIP suffers primarily L1 misses which almost all will hit L2 and the helper threads are primarily basic triggered. Therefore, the main thread rarely spawn any helper thread, and for those instance of spawning, the helper thread is not timely enough to be of any use.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]				
	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

[redacted]				
	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

[redacted]				
	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

[redacted]				
	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

	[redacted]				
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]					
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]					
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]					
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]